Dynamic list scheduling of threads on clusters

G. G. H. Cavalheiro, E. D. Benitez,
D. S. Peranconi, E. Moschetta

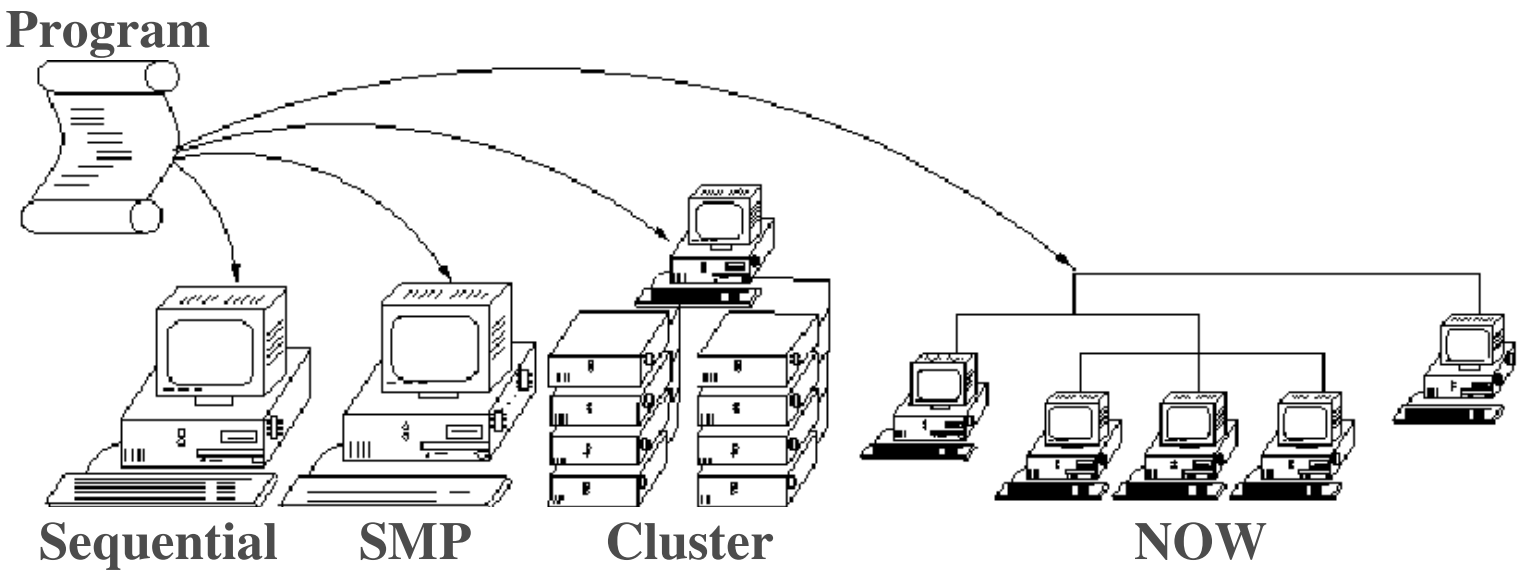
Universidade do Vale do Rio dos Sinos
Programa Interdisciplinar de Pós Graduação em Computação Aplicada

Overview

- **Introduction**
- **Anahy**
  - Task and synchronizations
  - Programming interface
  - Scheduling strategy
- **Handling a Graph of Tasks**
  - Visualizing an execution
- **Some Performances**
- **The Future of Anahy**

Introduction

- **Performance portability**



  - The concurrency of an application can be described regardless of hardware resources

Introduction

- **Performance portability**
- **Concurrency**
  - Depends on application characteristics
  - Can be identified by a specialist on the application
- **Parallelism**
  - Depends on hardware
  - A specialist on applications is not necessarily an specialist in parallel programming

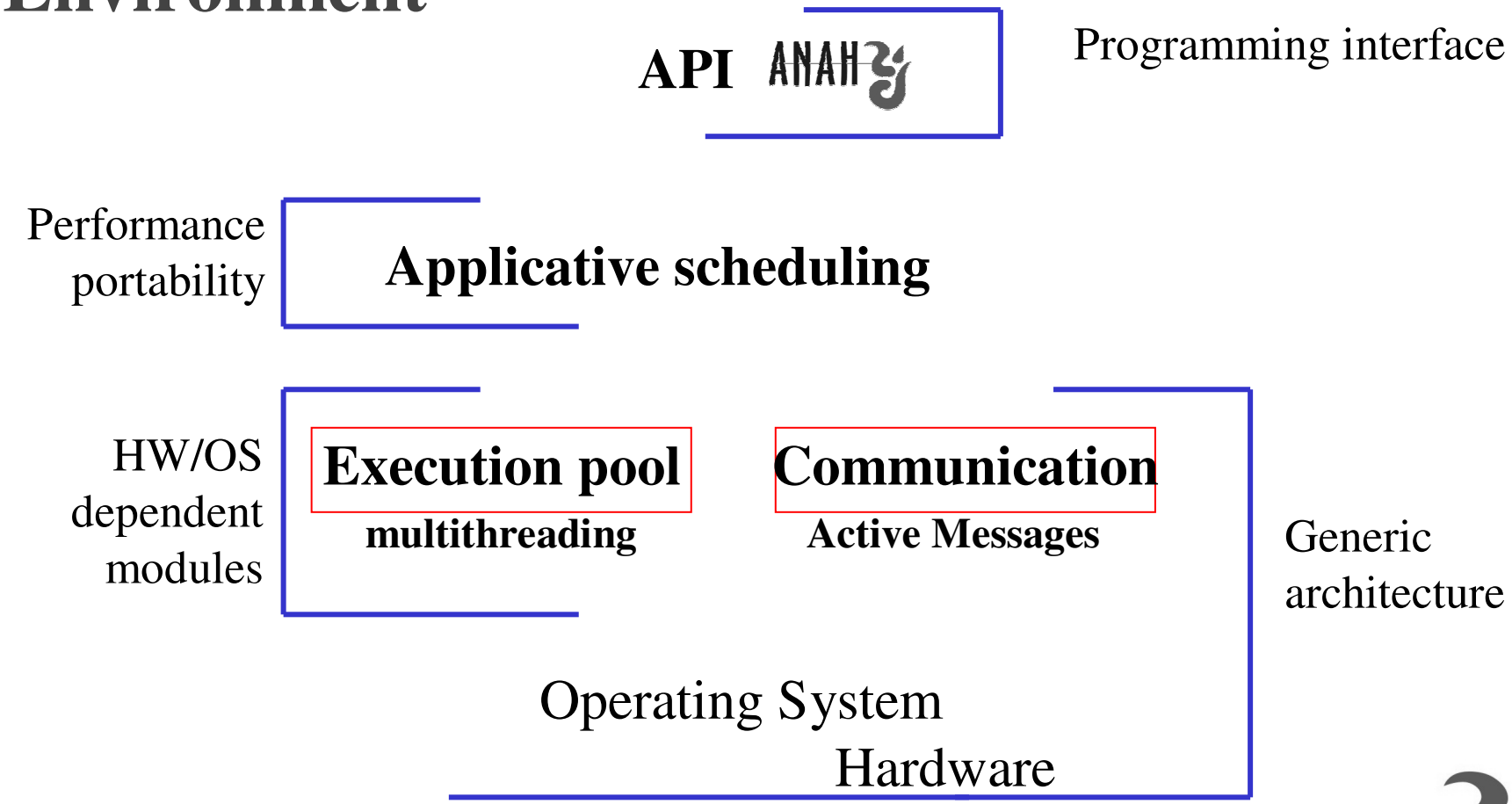
Concurrency >> Parallelism

Introduction

- **Performance portability**
- **Our approach:**
  - Dissociate programming of execution
- **Our proposal:**
  - ANAHY
- **Our mechanisms:**
  - Scheduling and dataflow control achieved at run time

Anahy

- **Environment**

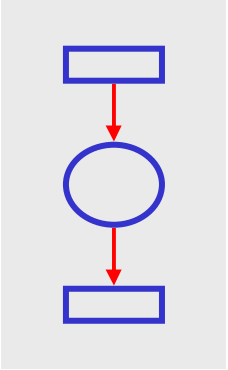


UNISINOS

Anahy

Task and Synchronization

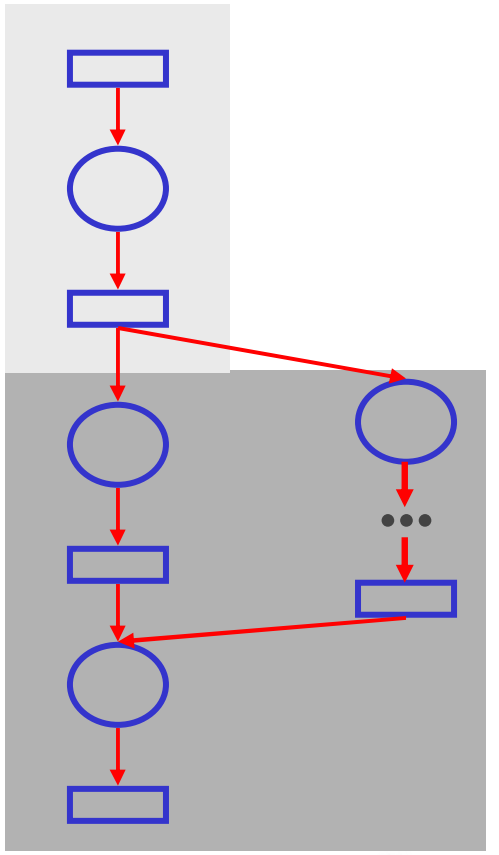
- A ***task*** defines a sequence of instructions and two set of data: input and output data;
- The synchronization between tasks are guaranteed by accesses to the data



Anahy

Task and Synchronization

- A *task* defines a sequence of instructions and two set of data: input and output data;
- The *synchronization* between tasks are guaranteed by accesses to the data



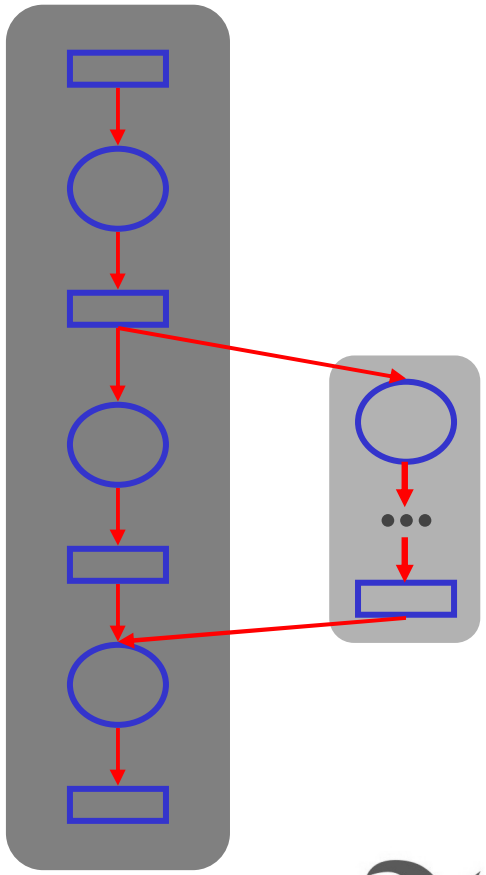
Large amount of concurrency

↳ **large amount of synchronizations**

Anahy

Task and Synchronization

- A *task* defines a sequence of instructions and two set of data: input and output data;
- The *synchronization* between tasks are guaranteed by accesses to the data



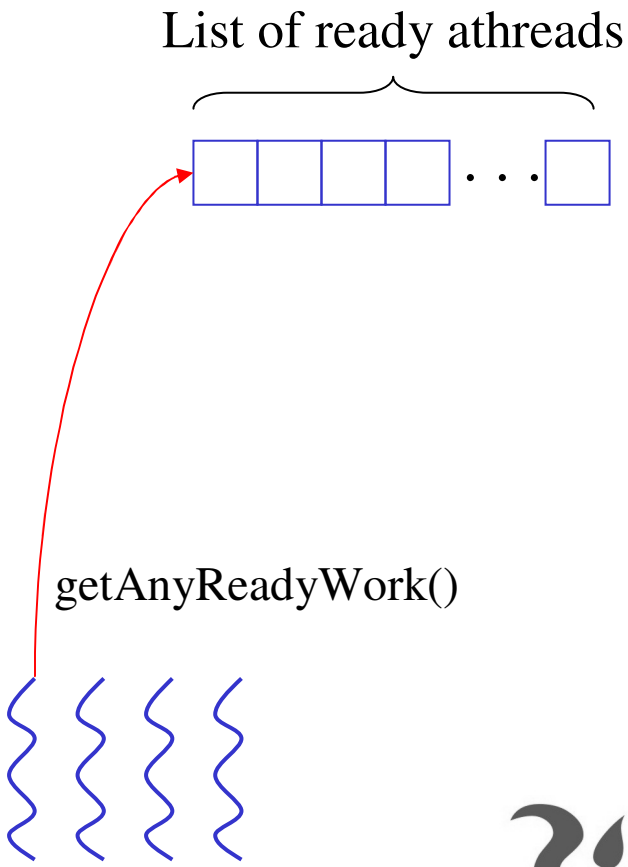
Large amount of concurrency

↳ **large amount of synchronizations**

Coarse scheduling unity: athread

Anahy

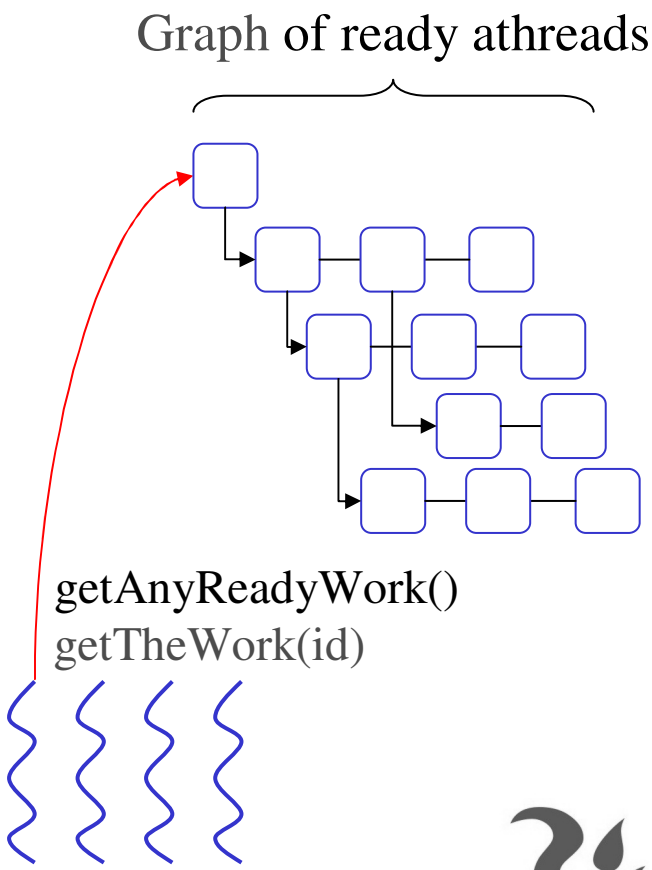
- Execution pool
  - A set of *system* threads is responsible for executing the athreads
  - Each system thread is called VP
  - Strategy:
    - A VP can chose a specific athread to execute



UNISINOS

Anahy

- Execution pool
  - A set of *system* threads is responsible for executing the athreads
  - Each system thread is called VP
  - Strategy:
    - A VP can chose a specific athread to execute
    - The list of ready works is organized as a graph of dependencies



UNISINOS

Anahy

- Programming Interface
- Creation

```
int athread_create( athread_t *th,
                    athread_attr_t *attrib,
                    void *(*func) (void *),
                    void *in );
```

- Synchronization

```
int athread_join( athread_t th, void **res );
```

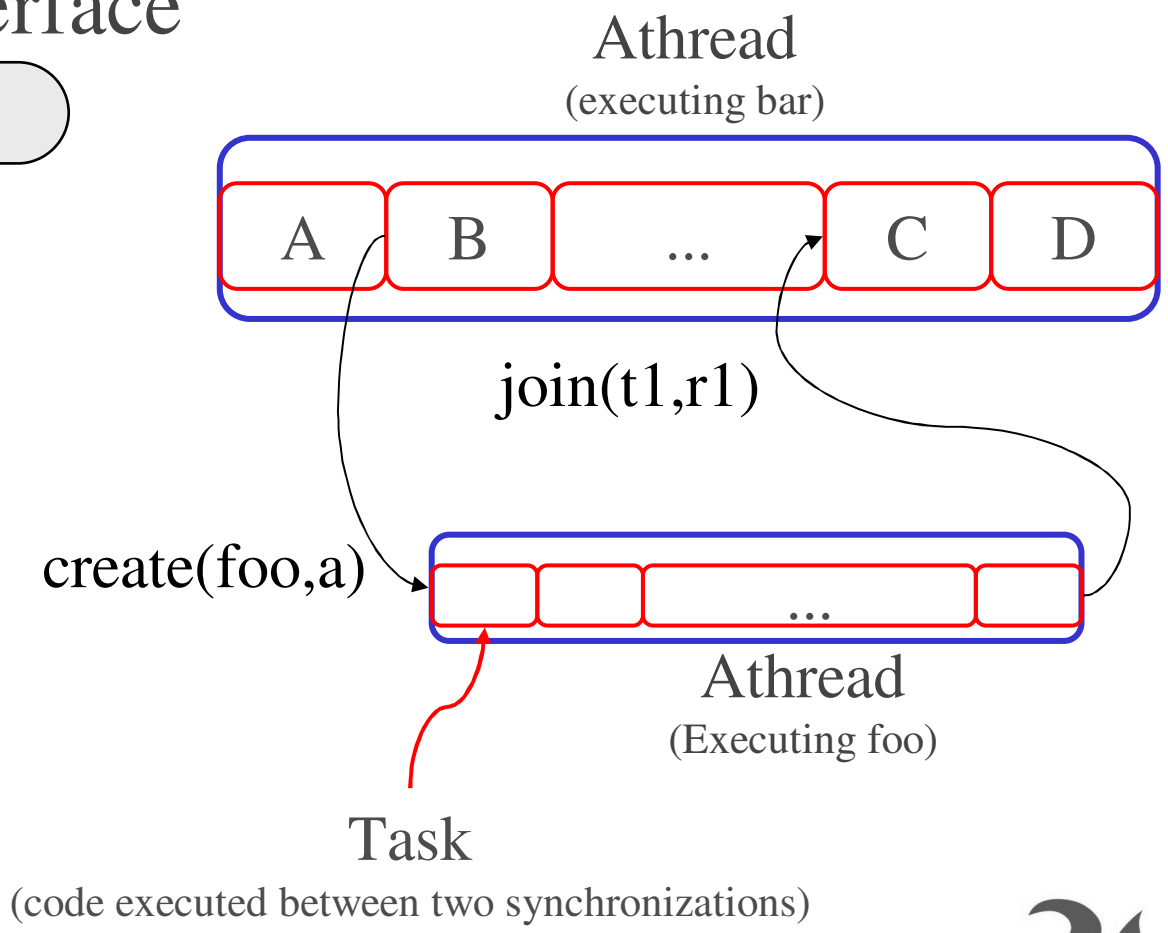
- Athread code

```
void *foo( void *in ) {
  ...
  return out;
}
```


Anahy

- Programming Interface

```
void* foo(void* x) {
 ...
}
void* bar(void* p) {
 Task_A
 t1 = create(foo,a);
 Task_B
 t2 = create(fuu,b);
 ...
 join(t1,r1)
 Task_C
 join(t2,r2)
 Task_D
 return &something;
}
```



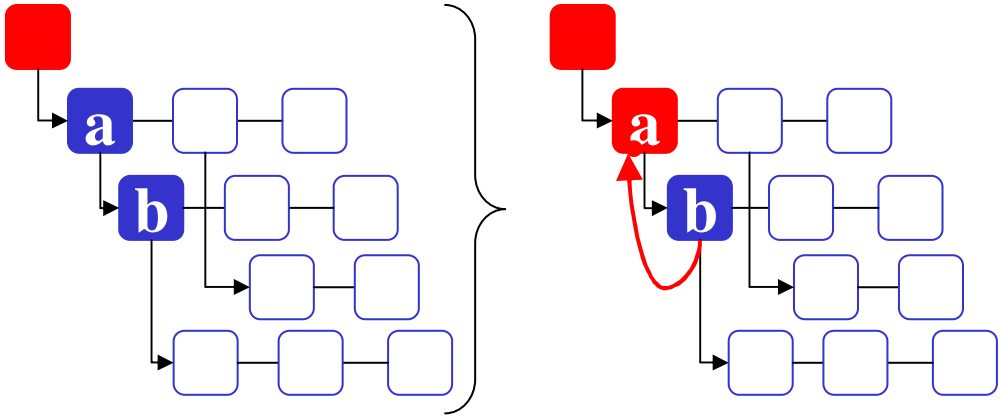
UNISINOS

Anahy

- Scheduling
- List scheduling
  - Blind strategy
    - Explosion on concurrency or memory
- Scheduling heuristics
  - Different searches on the graph
- Applied:
  - When a VP becomes idle and request for work
  - When is executed a join operation

Handling graph of tasks

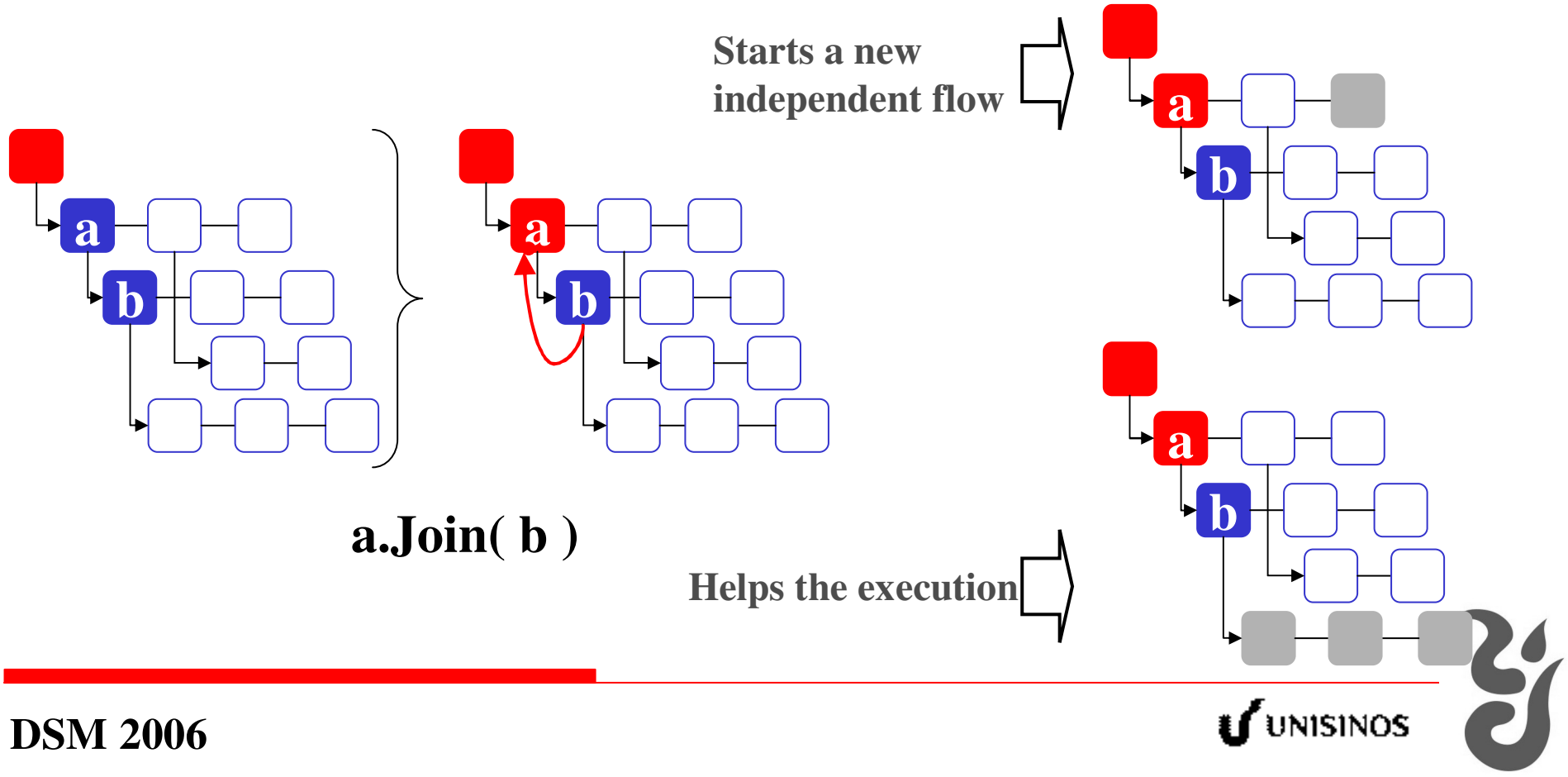
- Search an athread on the graph:
  - athread_t* SearchFrom(from, direction, orientation, axis)



a.Join(b)

Handling graph of tasks

- Search an athread on the graph:
  - athread_t* SearchFrom(from, direction, orientation, axis)



UNISINOS

Handling graph of tasks

- Examples
  - SearchFrom(current, ROOT, LEFT, VERT)
    - returns the next athread ready in the sub-graph having current as root (left-to-right, high priority on deep nodes)
  - SearchFrom(NULL, TOP, RIGHT, HORIZ)
    - returns the next athread ready in the graph from the first node of the graph (right-to-left, high priority on high nodes).
  - SearchFrom(jid, ROOT, RIGHT, HORIZ)
    - returns the next athread ready in the sub-graph having jid as root (right-to-left, high priority on the higest athread in the sub-graph).

Handling graph of tasks

- Visual example
  - Recursive program:

```
void* tree( void* n ) {
 if( n > 2 ) {
   t1 = create( tree, *n-1 );
   t2 = create( tree, *n-2 );
   doSomething( ... );
   join(t1,&r1);
   join(t2,&r2);
 }
 else doSomething( ... );
 return &something;
}
```

  - VP idle:
    - searches the last created in the highest level

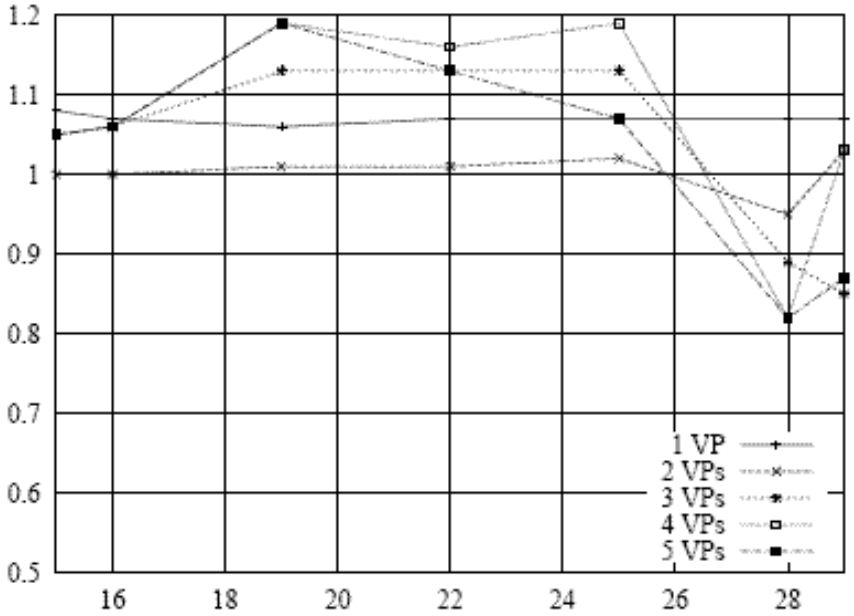
      SearchFrom(NULL, TOP, RIGHT, HORIZ)
  - athread blocked in a join:
    - searches a ready athread from jid

      SearchFrom(jid, HERE, RIGHT, HORIZ)

Performance

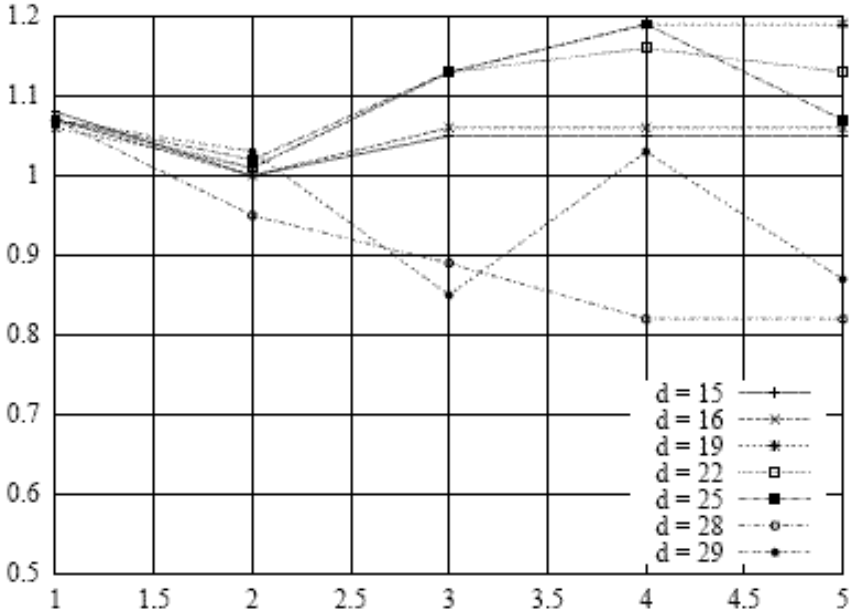
High Parallel Application

Ratio: Cilk / Anahy on a dual-processor



Depth

Concurrency level on the program



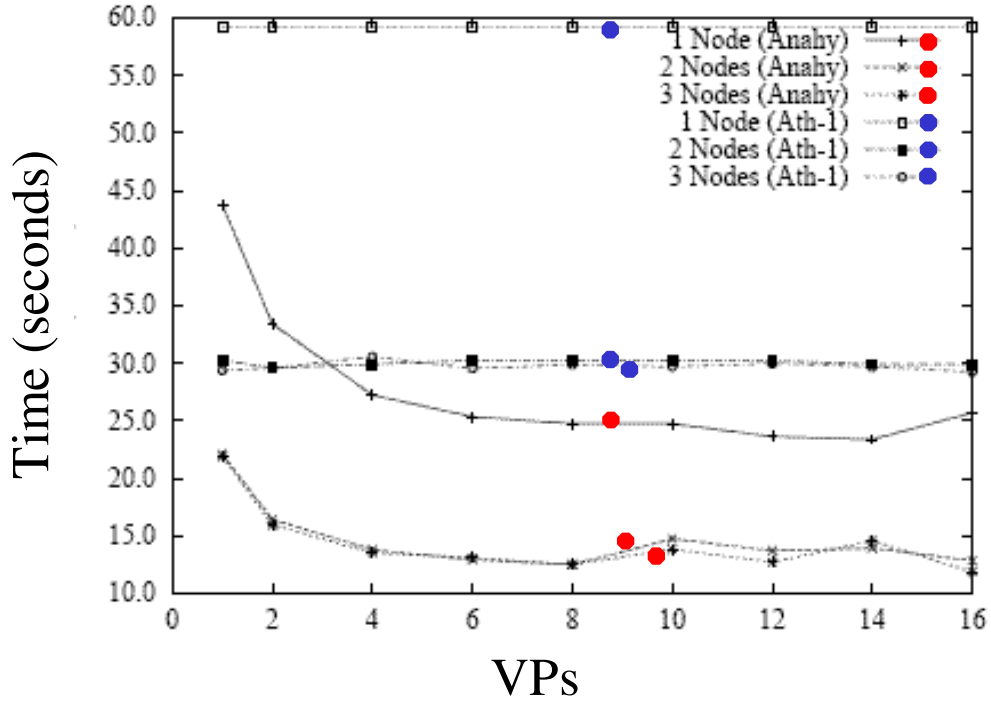
VPs

Parallel execution support

Performance

High Parallel Application

Execution times: Athapascan-1 *x* Anahy on a cluster



Parallel execution support per node

The future of Anahy

- Current work
  - Distributed version
  - Real applications
    - Dynamic programming
    - Metabolic cellular network
    - Crowd simulation
- Next
  - Scheduling strategies
- Next++
  - Other Pthreads synchronization mechanisms
    - Mutex, condition variables

Dynamic list scheduling of threads on clusters

G. G. H. Cavalheiro, E. D. Benitez,
D. S. Peranconi, E. Moschetta

gersonc@anahy.org, anahy@anahy.org

.org

UNISINOS